DTN-based Content Storage and Retrieval

Jörg Ott
Helsinki University of Technology (TKK)
Networking Laboratory
jo@netlab.tkk.fi

Mikko Juhani Pitkänen
Helsinki Institute of Physics
Technology Programme
pitkanen@mail.cern.ch

Abstract

Delay-tolerant networking (DTN) enables nodes to communicate by means of asynchronous messaging without the need for an end-to-end path. Suitably designed application protocols may operate in DTNs by minimizing end-to-end interactions and using self-contained messages for communication. The store-carry-and-forward operation and message replication of many DTN routing protocols may yield multiple copies of messages spread across many nodes for an extended period of time. We leverage these properties for application support in (mobile) intermediate DTN nodes which act as ad-hoc routers. We add explicit application hints to messages that are visible to each node, allowing them, e.g., to cache content, act as distributed storage, or perform application-specific forwarding.

1 Introduction

Powerful personal devices enable information access (mostly web), interpersonal communication (email, messaging), and content generation (photos, videos, blogs) for mobile users. They offer rich storage capacity and several wireless interfaces (WLAN, Bluetooth, cellular) to communicate via infrastructure networks but also directly with one another. Nevertheless, communication is focused on infrastructure access which imposes several limitations: 1) While the devices can be used anywhere, infrastructure networks are not ubiquitous so that communication may be disrupted for some time. 2) If infrastructure is available, its utilization may be suboptimal (and costly): as each user interacts individually, the same information may cross the wireless link repeatedly. 3) With the Internet as the hub for information exchange, proximity of nodes cannot be exploited to favor direct communication between "adjacent" nodes. This limits the *effective* communication capacity and may prevent obtaining the desired service.

Issue 1 can be addressed by ad-hoc networking among mobile nodes using Delay-tolerant networking [4] with message-based asynchronous store-carry-and-forward communication which does not require end-to-end paths between peers. This may allow communication even in environments with low node density [13], provided that application protocols are adapted accordingly [11]. Cooperative caching [26] may help mitigating issue 2; grid storage and peer-to-peer technology have been used to replace centralized systems by distributed approaches, (partly) addressing issue 3. However, they maintain their own infrastructure nodes (even if built up from peers) and require a certain degree of connectivity and stability—conditions which may not hold in sparse ad-hoc networks.

In this paper, we assume DTN-based messaging as the basis for communication. With DTN, application protocols do not rely on interactive protocol exchanges so that protocol messages are assumed to be semantically self-contained. Each application data unit (ADU), e.g., an email or the contents of a web page, is carried in one message. This means that both mobile and stationary intermediate nodes—which we refer to as bundle routers—will typically operate only on complete ADUs rather than individual IP packets. Because of the store-carry-and-forward nature of DTNs, these ADUs may remain *stored* within a DTN router's queue for an extended period of time. Moreover, copies of the ADU may be *replicated* across several nodes—depending on the routing protocol is use.

These observations motivate using (mobile) DTN nodes for temporary storage and retrieval of application data [14]. In this paper, we explore using the bundles (=ADUs) queued in DTN routers for caching and for distributed storage, leaving further application support for further study. In section 2, we briefly review related work. We introduce application scenarios in section 3, outline application support in DTN routers and in the bundle protocol [20] in section 4, and present simulation results for data caching and storage in section 5. We discuss open issues and conclude this paper in sections 6 and 7.

2 Related Work

Delay-tolerant Networking has been applied to various challenging networking environments, including space communications, sparse sensornets, and opportunistic mobile ad-hoc networks. The DTN architecture [1] proposes a *bundle layer* to span different (inter)networks. Messages—*bundles*—of arbitrary size are forwarded hop-by-hop between DTN nodes (*bundle routers*) as best-effort traffic.

They have a finite TTL. *Custody transfer* provides reliable transmission through persistent hop-by-hop forwarding. Control messages are used to inform a sender or custody router about message forwarding progress, end-to-end delivery, or failures [20].

Numerous DTN routing schemes have been proposed, such as simple message replication (not just) for mobile environments and the use of multiple paths to increase delivery probability [6] [10]. Various algorithms for creating, forwarding, and purging copies of messages have been investigated. Furthermore, erasure and network coding techniques have been suggested [24] [25] [5] to reduce traffic. The DTN queuing/forwarding properties and the resulting propagation delays have been studied for multicasting [27] and exploited for broadcasting [7], with messages spreading hop-by-hop using DTN as the basis for content distribution. Management of queued DTN bundles has been studied for congestion control with custody transfer [21].

Besides web caching, distributed data storage has been dealt with in peer-to-peer systems which store data across client nodes (with or without replication) and enable distributed location and retrieval (e.g., [22] [18]). Grid systems have been used to create distributed storage space [8], also using erasure coding-based for scalable storage [16]. However, in most cases, individual nodes are assumed to be well connected. OceanStore [3] has investigated storage access for weakly connected nodes and others (e.g., [2]) have studied applying peer-to-peer overlays in mobile ad-hoc networks.

In contrast to the approaches above, we assume DTN-based communication, allowing to tolerate disconnections and operating on large data bundles identical to ADUs. We embed *application hints* in each bundle to leverage the storage and propagation properties of DTNs to support applications. These hints may even be used to *actively influence* bundle queuing and forwarding. This somewhat resembles the integration of forwarding and (limited) application-specific processing by means of mobile code in *Active Networks* [23]. However, the typically larger bundle size reduces the frequency at which processing needs to be performed and the overhead of carrying the hints (active networks: code) [14]. Moreover, as bundles usually contain entire ADUs (rather than just fragments as packets often do), application-specific operations are simplified: all the data is available at once and decisions whether and which supportive operations to perform can be based upon more complete context. This also allows more complex operations than QoS forwarding. In the following, we look at requesting, retrieving, and caching self-contained ADUs, but also support their (temporary) distributed storage.

3 Application Scenarios

We assume application protocols designed or adapted for use with DTNs (as proposed for HTTP [12] or email [15]), that the contents is of relevance to several users as (e.g., web pages, newsgroups), and that protocol operations follow a request-response (like HTTP) or publish-subscribe (like NetNews) paradigm. We look at two partly overlapping scenarios: resource retrieval with caching and distributed resource storage in conjunction with retrieval. We derive common requirements for DTN support, focusing on the protocol functionality and deferring security aspects to future work (see also section 7).

3.1 Content Retrieval and Caching

A mobile user wants to *retrieve* resources available on the Internet which are represented as web pages. A web page request from a requester R_1 contains an HTTP GET message in which the Request-URI and possibly the Host: header field identify the resource U sought. The request $req_1(U)$ travels from R_1 via intermediate nodes N_i and an access link to the fixed Internet and finally reaches the origin server. The latter returns a response $rsp_1(U)$ containing the complete resource U plus a set of headers describing attributes of the resource (e.g., modification date, cache control, lifetime)—or an error.

Depending on the routing algorithm, both $req_1(U)$ and $rsp_1(U)$ may be replicated repeatedly and traverse several nodes N_i along different paths so that multiple copies may exist and reach server and client. A copy is kept on a DTN router until forwarding requirements by the routing algorithm are satisfied, their respective DTN TTL expires, or a queue replacement algorithm decides to drop a bundle.

If, at a later point in time, another client R_2 also issues a request $req_2(U)$ for U, $req_2(U)$ may traverse a node N_j that still has a copy of $rsp_1(U)$ queued.[1] In this case, N_j acts as an *implicit cache*, creates $rsp_2(U)$ from its queued $rsp_1(U)$, and replies to R_2 The prerequisites are that N_j is able to identify the resource U to match $req_2(U)$ and $rsp_1(U)$ and to determine whether the freshness requirements from the request are satisfied. If response bundles are stored even longer than necessary for DTN delivery (e.g., for their application lifetime as opposed to their DTN TTL), a cooperative cache can be built among mobile users from DTN queuing mechanisms.

3.2 Content Storage and Retrieval

We extend this scenario to also cover *store* operations. Assume another mobile user S who wants to create web-based resources, e.g., to maintain a blog about her trip and archive her digital pictures, making the blog and a subset of the pictures also available to other travelers in the area. S uses DTN to reach a server in the fixed Internet that accepts

[1] This depends on many factors including the proximity of R_1 and R_2, the motion of nodes N_i, the routing protocol, the delay between req_1 and req_2, and the bundle TTL. Given the positive experience with web caching experience for co-located nodes and assuming that DTN bundles will first spread locally, these dependencies appear acceptable.

the contents, generates blog entries, and archives images; contents stored by the server are acknowledged end-to-end so that the traveler knows she can delete the data locally.

The storage request $srq(U)$ containing the resource U propagates towards the fixed Internet, again, being stored, forwarded, and replicated along multiple paths on nodes N_i. While queued in N_j, this intermediate node can make U available to a requester R who is interested in U and issues a request $req(U)$. N_j has to identify the local copy of U (which is embedded in $srq(U)$) and generate the corresponding response $rsp(U)$ for R from $srq(U)$.

Effectively, all nodes N_i maintaining a copy of U at a given point in time form a distributed storage in a DTN. Setting the DTN TTL equal to the application lifetime of U or the desired period of local availability (instead of the expected time to reach the fixed server) and assuming sufficient queue capacity, allows the resource to be kept available among the mobile nodes until its expiry and thus enables retrieval by other (co-located) nodes R without having to contact the infrastructure network. This can also assist in sharing resources even if no infrastructure is available at all: rather than addressing ADUs to a fixed target, a new address type could be used to keep the resources around (e.g., some variant of multicast or anycast).

With $R = S$, the above mechanism may also serve as a distributed backup storage for S if multiple copies (or other forms of redundancy) of the bundle $srq(U)$ are generated so that its contents can likely be retrieved later and help prevent information loss in case of hardware loss or failure.[2]

In all cases, queuing is more relied upon which motivates resource versioning and explicit *cleanup* bundles for purging contents outdated or no longer needed.

3.3 Common Requirements

In the above scenarios, intermediate DTN nodes perform application-supportive processing in addition to the usual store-carry-and-forward operation. This may include matching requests and responses, generating responses, and modifying their routing and buffer management. For such processing, bundles need to provide additional information:

Application protocol: Needed to perform proper resource matching, to allow for selective support (e.g., if a node only wants to or can support certain protocols), and to generate proper responses to retrieval requests. Processing steps requiring further details can parse the ADU contents and react accordingly, e.g., interpret conditional requests as known from HTTP.

Resource: The resource carried in a bundle or asked for needs to be identifiable for matching operations. For more advanced processing, the resource's *content type* and encoding and possibly other parameters might be provided.

[2]Temporarily moving custody bundles to another node if local buffer space is scarce has been suggested for buffer management at the DTN layer [21].

Operation type: Independent of the application protocol, the general class of operation should be indicated to allow for minimal support even if the application details are not understood: is this a request for retrieving a resource, an error message, or a bundle containing a resource? This allows different processing, storage, and routing policies.

Lifetime: If a resource is contained in the message, its application layer lifetime should be made explicit, so that keeping and discarding bundle may be adapted accordingly and the DTN TTL does not need to be overloaded.

We suggest an additional bundle protocol block [20], *Application-Hints*, to indicate that a bundle seeks advanced treatment and to convey: the resource URI (and thus the application protocol), the operation indicator, the resource lifetime, and optional resource-specific parameters. Such a block can be encoded in a few bytes plus the URI and optionally the media type and thus will not increase the bundle size significantly. In case of bundle fragmentation, this block needs to be copied into every fragment.

4 Application-aware DTN Routers

Figure 1 shows a simplified diagram of an intermediate DTN node (=router) in an ad-hoc environment. A bundle comprising a DTN header "H" and a payload arrives on an incoming interface on the left. After some validity checking (e.g., authentication, TTL) it is put into a queue, subject to the local queue management policy which may discard the bundle itself or cause other bundles to be dropped (e.g., based upon drop-tail, RED, oldest bundle, first expiry).

In an opportunistic routing environment like an ad-hoc DTN, forwarding decisions are not taken upon arrival but rather when one or more contacts become available via the outgoing interface(s). Each contact provides information to the routing logic: that the contact is now available and possibly other routing metrics. The updated routing information is then used for forwarding (which bundles to transmit to the contact) and scheduling (in which order) decisions.

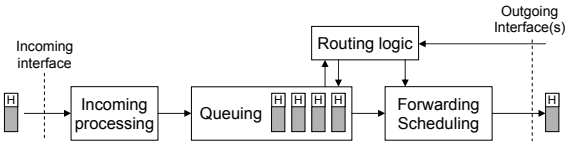


Figure 1. Simplified view of a DTN router

The routing scheme determines if (and for how long) the transmitted bundles are kept in the queue for further contact opportunities or are discarded (which happens latest when their DTN TTL expires). The forwarding algorithm may use partial or full replication of bundles at different stages to increase their delivery probability: for example, only the sender may create copies, only n copies in total may be

created, or bundles may be forwarded only a finite number of times (before the respective destination node is met).

Routing, forwarding, and queue management use bundle headers. They indicate parameters for transmission from the source to the destination—but do not necessarily reflect properties of the bundle contents. In the following, we introduce two extension steps to bundle routers based upon the *Application-Hints* header proposed above: the passive mode of operation only uses hints for locating contents and generating replies while the active mode also influences bundle routing, forwarding and/or queuing decisions.

4.1 Passive Operation

Passive support does not alter the basic bundle processing functions of a DTN router. As depicted in figure 2a, an *application-aware processing (AAP)* module gathers information upon every bundle event occurring in the router: it obtains information about newly arriving bundles (*info*) that contain an *Application-Hints* header "A" and learns when these bundles get *add*ed to and *delete*d from queues. By inspecting the information in the *Application-Hints*, the module obtains an overview of all presently queued bundles from an application perspective. This allows the AAP module to match a local copy of a resource to an incoming request (or an incoming resource to a queued request) and *retrieve* the resource from a queue[3]. If the stored resource has the form of a valid reply (e.g., an HTTP 200 OK response), a copy of it may be retargeted to the requester. If a response to a *retrieve* request needs to be adapted or a resource needs to be extracted from a *store* request, an application-specific component is used to generate a *reply* bundle and enqueue it for delivery to the requester.

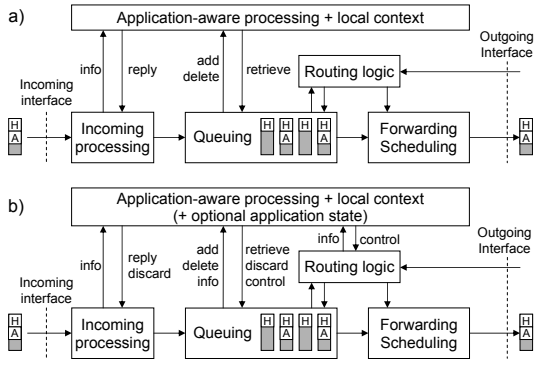


Figure 2. Passive (a) and active (b) support for applications integrated in a DTN router

[3]In principle, an AAP module could also duplicate selected bundles in some local storage (rather than operating on the DTN router queues); but sharing one copy of the potentially large bundles reduces the memory requirements.

4.2 Active Operation

In the passive mode, the AAP module cannot influence decisions of the local router[4] which has the benefit of minimizing interference with the "underlying" routing protocols. There are, however, circumstances under which such influence may be desirable: In a simple case, an AAP module serving repeated requests for a resource from the local queue may want to prevent the corresponding bundle from being purged and instead keep it until its application-level lifetime expires. Similarly, an AAP module may speed up deletion of bundles no longer needed or may treat bundles differently depending on their operation type. It may also supply routing or forwarding hints per application protocol and thus enable different routing schemes in parallel—which could be used to construct overlays as part of the regular routing mechanisms. Finally, a mobile node may exercise better control of its local capacity by selectively forwarding bundles for some applications (which the user is interested in) while not accepting them for others.

To realize this active control, further interfaces are added (figure 2b): in addition to those discussed above, the AAP module may *discard* incoming bundles right away and queued ones at a later time. It may also modify bundle attributes relevant to queue management (*control*) to influence local buffer management. Finally, the AAP module interacts directly with the routing logic: it receives information (*info*) about contacts and routing updates and is able to supply routing, forwarding, and scheduling preferences via the *control* interface.[5]

These operations allow an AAP module to speed up spreading of some bundles while delaying or preventing further dissemination of others. Effectively, the AAP module can thus control the degree of replication, the choice of contacts for forwarding, or the amount of erasure coding being applied to a particular bundle to match application needs. For example, if DTN nodes serve as storage for certain resources according to scenario 2 above, a higher degree of replication and a longer lifetime are desirable than for simple retrieval requests. However, the active AAP modules need to be conservative in their influence to avoid unexpected feature interactions with DTN layer routing or other application functions.

5 Some Simulation Results

In this section, we investigate the performance implications when applying *mostly passive* router support to content retrieval with caching and distributed content storage. We have carried out simulations for 1) a challenging artificial setup with fixed dense nodes and highly dynamic links yielding frequent contacts and disruptions and 2) a sparse

[4]Which also prevents deleting a request bundle that was just replied to.

[5]The *dtnd* of the DTN reference implementation provides an external router control interface used, e.g., for prototyping routing protocols.

mobile environment based upon real-world traces with rare contacts.

We have extended a Java-based DTN simulator [6] to include bundle transfers with duplicate detection based on bundle fingerprints, URIs for bundle content identification, and caching/retrieval mechanisms with a configurable probability. We do not use fragmentation: incomplete transfers are considered failed. We use a variant of flooding for forwarding which is less aggressive than epidemic routing: bundles are stored in a FIFO queue and are forwarded to all contacts available at the same time. Messages are deleted from the queue if at least one forwarding was successful and are kept otherwise to be retried until their TTL expires.

5.1 Challenging Artificial Scenario

Figure 3 depicts the network topology used in the first two simulations. A simple topology was chosen to avoid biases and very fast link state transitions were employed to create a challenging environment. The clients (C) reside on one edge of a 10x10 network and the servers (S) on the opposite side. Nodes connect to their neighbors by 802.11 WLAN links at 11 Mbit/s with delays of 100 ms. The link behavior follows a pattern with the up- and downtime exponentially distributed with a 16 s and 4 s mean, respectively. These times are shorter than likely seen in reality but allow us to observe protocol behavior with highly intermittent connectivity and frequent bundle transmission failures (which we expect to be common).

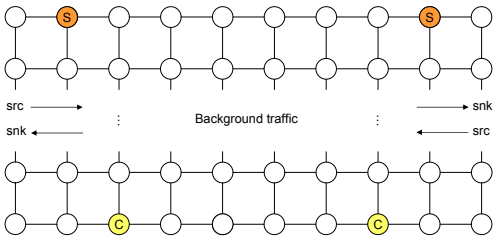


Figure 3. Simple simulation grid topology

The clients send requests independently every 3 minutes: 70% of the requested resources are chosen from 4 URIs of core interest, the remaining 30% from a pool of 400 URIs. Requests and responses time out after 20 s, but the intermediate nodes may cache responses for 20 min if cache space is available. Each node has a DTN queue size of 500 MB and additional 100 MB for caching. We run simulations over 27 hours of time with background traffic (randomly sized messages between two node pairs, *src* to *snk*) causing frequent overflows in the forwarding buffers. All simulations are repeated 3 times with different random seeds of which we plot the mean values.

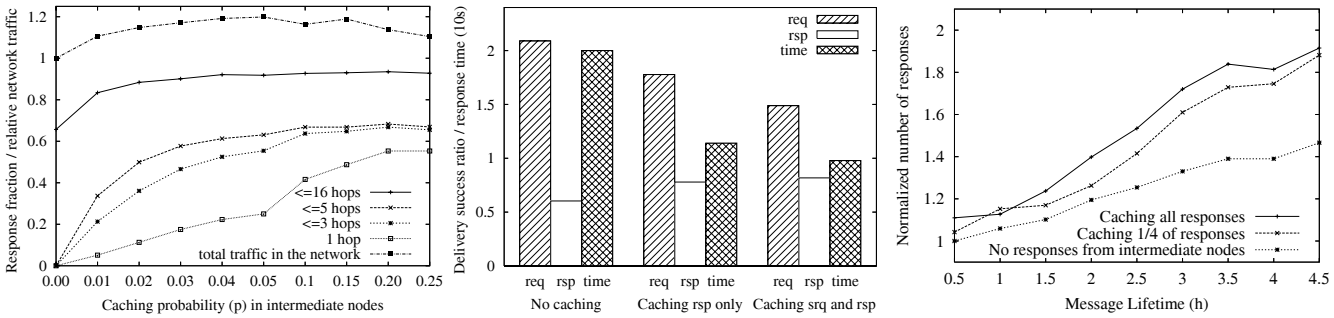
Figure 4a illustrates the effect of caching bundles in nodes even after successful forwarding. The caching decision is taken independently for each message with the probability p (X axis). On the Y axis, the *response fraction* shows the likelihood that a client receives the response from a cache at most n hops away. With increasing p the fraction of such response grows, particularly for small n and p. This underlines the positive impact of caching. Despite the short bundle TTL, up to 10% of the responses are returned from the nodes' DTN queue, the remaining 90% from the cache part with longer storage time. We also plot the total traffic relative to the amount without caching: There is a slight increase as requests get fanned out and multiple responses are returned which is offset with increasing p by the shorter distances that requests (and responses) travel. Overall, we observe that already small p lead to significant performance improvements.

Figure 4b shows how different ways of caching perform if contents is stored (*srq*) by some mobile nodes and retrieved (*req, rsp*) by others: With *no caching*, all storage and retrieval requests and responses have to travel to the server and back. For *caching rsp only*, responses may be cached, and *caching srq and rsp* also caches store requests heading towards the servers. If caching is enabled we use $p = 0.1$ from figure 4a and the same parameters as above except that now the clients issue store requests in 50% of the cases. The first bar plots the fraction of *retrieval* requests for a bundle that reach the server; values larger than 1.0 indicate duplicates. The second bar indicates the fraction of requests for which the clients receive a response. The third bar shows the mean time until a response for the requested resource is received (in 10 s units). Across all three setups (not shown), 60% of all *srq* bundles make it to the server—the upper bound for infrastructure-based retrieval of contents originated by mobile users.

We can clearly see how the caching for *rsp* bundles reduces the number of *req*s forwarded to the servers and the response time because many requests are already responded by the intermediate nodes and not forwarded further. Both decrease further when intermediate nodes also cache *srq* messages and use them reply to requests. At the same time, the fraction of successful requests increases from some 60% without caching to more than 80% when using active distributed storage. Hence, with DTN-based caching and storage, the proximity of nodes providing and retrieving contents may be exploited to increase efficiency–and even help fulfilling requests that would fail when relying on infrastructure servers.

5.2 Sparse Mobile Scenario: Haggle

We have also investigated the effect of caching using realistic contact traces as obtained from an experiment in the Haggle project [9] using the same node implementation as above. From the Haggle trace, we include only those nodes for which at least 30 contacts are reported (a total of 122 nodes) and use only iMotes or fixed nodes for



(a) Effect of caching probability

(b) Different caching alternatives

(c) Effect of caching in a mobile scenario

Figure 4. Simulation results for content aware retrieval in different DTN scenarios.

caching. We assume Bluetooth EDR connections with a capacity of 2.1 Mbit/s and used again 500 MB forwarding and 100 MB additional caching buffer. Six fixed and four mobile nodes served as gateways to the Internet to access fixed servers. We randomly requested files of sizes 600, 1200, and 1800 KB (which seems realistic for a log containing photos) from two mobile and three stationary nodes. Due to the limited connectivity, no forwarding buffer overflows were observed during our simulations.

Figure 4c illustrates the impact of caching and the message TTL (the caching lifetime is set to 15 times the message TTL, the x-axis indicates the message TTL) on the number of responses received. We normalize the results to the number of responses received without caching at a TTL of 30 min (39 responses) and plot the responses with different caching probabilities. We observe 1) that caching increases the response probability even in sparsely connected scenarios, 2) that the increasing message and caching lifetime yields better performance increase with caching (roughly a factor of two), and 3) that, again, limited caching already yields noticeable gain. Further simulations (not depicted) have shown that most responses come from three hops or less in distance which further emphasizes the achievable performance gain from caching. With caching, the response time to bundle requests (not shown in the graph) ranges from 1.5 to 3 hours, which appears reasonable for our travel scenario where shared contents can be obtained without infrastructure use. Further investigations will have to show how well proximity can be exploited when also realizing temporary distributed storage.

6 Discussion

Application support in intermediaries raises the issue of how much knowledge is needed (and acceptable!) in these nodes and how much support can be provided without understanding the application protocols. All actions that do not create bundles or alter their contents but only modify bundle treatment or replicate existing ones can operate independently of the applications: they may perform caching, influence forwarding or queuing, the degree of replication, and the bundle lifetime.

To generate responses or perform other protocol operations, nodes must know how to handle the application protocol. This functionality may be hardcoded into DTN nodes or provided via application-specific plug-ins. Supporting a few widely used protocols in this way may already suffice to cover large fractions of the traffic. However, protocols evolve over time. For more flexibility, minimal mobile code (as in Active Networks [23] or signaling compression [17]) could be sent along with a *retrieve* request and contain instructions for creating the response from a *store* request.

Since the enhanced DTN router modules operate probabilistically and the basic routing and forwarding operations do not depend on such support, applications will still work even if their bundles are not recognized by some or all of the nodes—which also allows for incremental deployment and does not require agreement among nodes on which protocols to support. However, while this opportunistic operation avoids that applications break if no support is available, actively influencing the network may lead to a loss of transparency and hence requires careful consideration [19].

7 Conclusion

In this paper, we have suggested to leverage two DTN properties—the transmission of ADUs in self-contained bundles and their queuing in intermediate nodes for some time—to provide support for applications. Our approach allows generic functions such as bundle routing to be performed differently per application, operation, or resource, but particularly enables application support by means of caching or distributed storage. The proposed extensions are incrementally deployable and robust as the regular network operation does not depend on particular or any nodes offering such support at all. Our initial simulations show that even a small contribution from supportive DTN nodes suffices to increase application performance.

Detailed application protocol analysis (and their redesign for DTNs) will yield further insights how much

functionality can (and should) be provided application-independently, whether and where mobile code can help, and when application-specific protocol engines are needed. Obviously, security issues require special consideration, including content authentication and integrity protection as well as privacy (e.g., for distributed storage) which also impacts application protocol design at large, as we have started discussing elsewhere [14] [11].

In our future work, we will address further setups, mobility models, and routing protocols as well as erasure coding techniques and investigate the implications for congestion control and scalability. Particularly the issue of feature interaction between application-specific control and the underlying DTN routing in heterogeneous environments requires further study.

8 Acknowledgements

This work was funded by the Academy of Finland in the DISTANCE project.

References

[1] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Network Architecture. RFC 4838, April 2007.

[2] F. Delmastro. From Pastry to CrossROAD: CROSS-layer Ring Overlay for AD hoc networks. In *Proceedings of the Percom Workshop on Mobile Peer-to-Peer*, March 2005.

[3] J. K. et al. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

[4] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proceedings of ACM SIGCOMM*, 2003.

[5] S. Jain, M. Demmer, R. Patra, and K. Fall. Using redundancy to cope with failures in a Delay Tolerant Network. In *Proceedings of ACM SIGCOMM*, 2005.

[6] S. Jain, K. Fall, and R. Patra. Routing in Delay Tolerant Networks. Proceedings of the ACM SIGCOMM, 2004.

[7] G. Karlsson, V. Lenders, and M. May. Delay-tolerant Broadcasting. In *ACM SIGCOMM CHANTS Workshop*, 2006.

[8] P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. File-based replica management. *Future Gener. Comput. Syst.*, 21(1):115–123, 2005.

[9] J. Leguay, A. Lindgren, J. Scott, T. Friedman, J. Crowcroft, and P. Hui. CRAWDAD trace set upmc/content/imote (v. 2006-11-17). Downloaded from http://crawdad.cs.dartmouth.edu/upmc/content/imote, Nov. 2006.

[10] A. Lindgren and A. Doria. Probabilistic Routing Protocol for Intermittently Connected Networks. Internet Draft draft-lindgren-dtnrg-prophet-02, Work in Progress, March 2006.

[11] J. Ott. Application Protocol Design Considerations for a Mobile Internet. In *Proceedings of the 1st ACM MobiArch Workshop*, 2006.

[12] J. Ott and D. Kutscher. Bundling the Web: HTTP over DTN. In *Proceedings of WNEPT 2006*, August 2006.

[13] J. Ott, D. Kutscher, and C. Dwertmann. Integrating DTN and MANET Routing. In *ACM SIGCOMM CHANTS Workshop*, 2006.

[14] J. Ott and M. J. Pitkänen. Application-aware DTN Routing. Technical Report, TKK Netlab, August 2006.

[15] A. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking Connectivity in Developing Nations. *IEEE Computer*, 37(1):78–83, January 2004.

[16] M. Pitkanen, J. Karppinen, and M. Swany. Erasure codes for increasing the availability of grid data storage. In *Proceedings of Workshop on Next-Generation Distributed Data Management*, 2006.

[17] R. Price, C. Bormann, J. Christoffersson, H. Hannu, Z. Liu, and J. Rosenberg. Signaling Compression (SigComp). RFC 3320, January 2003.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, 2001.

[19] D. P. Reed, J. H. Saltzer, and D. D. Clark. Active networking and end-to-end arguments. http://web.mit.edu/Saltzer/www/publications/endtoend/ANe2ecomment.html, 1998.

[20] K. Scott and S. Burleigh. Bundle Protocol Specification. Internet Draft draft-irtf-dtnrg-bundle-spec-08, Work in progress, December 2006.

[21] M. Seligman, K. Fall, and P. Mundur. Alternative Custodians for Congestion Control in Delay-tolerant Networks. In *ACM SIGCOMM CHANTS Workshop*, 2006.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, 2001.

[23] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. Proceedings of ACM SIGCOMM, 1996.

[24] Y. Wang, S. Jain, M. Martonosi, and K. Fall. Erasure Coding Based Routing for Opportunistic Networks. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[25] J. Widmer and J.-Y. L. Boudec. Network Coding for Efficient Communication in Extreme Networks. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[26] L. Yin and G. Cao. Supporting Cooperative Caching in Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 5(1):77–89, January 2006.

[27] W. Zhao, M. Ammar, and E. Zegura. Multicasting in Delay Tolerant Networks: Semantic Models and Routing Algorithms. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.